A distributed wheel sieve algorithm using Scheduling by Multiple Edge Reversal

Gabriel Paillard, Felipe Franca, Christian Lavault

A distributed wheel sieve algorithm using Scheduling by Multiple Edge Reversal

Gabriel A. L. Paillard[1], Felipe M. G. França[2]
and Christian Lavault[3]

[1] Universidade Federal do Ceará
Instituto Universidade Virtual
Av. Humberto Monte, s/n, bloco 901
Cep: 60440-554, Fortaleza - Ceará, Brazil
gabriel@virtual.ufc.br
[2] Universidade Federal do Rio de Janeiro
Programa de Engenharia de Sistemas e Computação
COPPE/UFRJ - Caixa Postal 68511
Cep: 21941-972, Rio de Janeiro - RJ, Brazil
felipe@cos.ufrj.br
[3] Université Paris 13, Sorbonne Paris Cité
Laboratoire d'Informatique de Paris Nord - LIPN
CNRS UMR 7030, F-93430 - Villetaneuse, France
lavault@lipn.univ-paris13.fr

Abstract. This paper presents a new distributed approach for generating all prime numbers in a given interval of integers. From Eratosthenes, who elaborated the first prime sieve (more than 2000 years ago), to the current generation of parallel computers, which have permitted to reach larger bounds on the interval or to obtain previous results in a shorter time, prime numbers generation still represents an attractive domain of research and plays a central role in cryptography. We propose a fully distributed algorithm for finding all primes in the interval $[2 \ldots, n]$, based on the *wheel sieve* and the SMER (*Scheduling by Multiple Edge Reversal*) multigraph dynamics. Given a multigraph $\mathcal{M}$ of arbitrary topology, having N nodes, an SMER-driven system is defined by the number of directed edges (arcs) between any two nodes of $\mathcal{M}$, and by the global period length of all "arc reversals" in $\mathcal{M}$. The new prime number generation method inherits the distributed and parallel nature of SMER and requires at most $n + \lfloor \sqrt{n} \rfloor$ time steps.
Keywords: Distributed Algorithms, Prime Numbers Generation, Wheel Sieve, Scheduling by Edge Reversal, Scheduling by Multiple Edge Reversal.

1 Introduction

This article takes up the generation of prime numbers smaller than a given bound n, by using the wheel sieve distributively. Wheel sieve algorithms can be very efficient to determine the primality of integers which belong to a given finite

interval $[2\ldots,n]$, for sufficiently large values of n and when the test of primality is carried out on all numbers of the interval. The paper designs a fully distributed wheel sieve algorithm using scheduling by multiple edge reversal (SMER).

The main purpose of a parallelization of such kind of algorithm is to increase the bounds of the generation of prime numbers, and to reach these bounds in a shorter execution time. The first parallelization of a sieve algorithm was realized in 1987 [4], who parallelized the sieve of Eratosthenes. This work was motivated by testing a new parallel machine (the *Flex/32*), because this kind of algorithm is ideal to test the performances of a new architecture (of a sequential or parallel machine) as a benchmark.

The sieve of Eratosthenes was the first prime sieving algorithm, and it consists in eliminating all non prime numbers in the interval $[2\ldots,n]$. First, the algorithm takes the first number of the interval and generates all its multiples (by adding its own value to himself), which are thus eliminated. The next (non eliminated) number is the one (the next prime number) which sieves the interval, and this process is pursued until all intervals has been sieved. Various parallelizations of this algorithm can be found, e.g. in [20, 21].

However, the main drawback of the practical sieve of Eratosthenes is clearly the fact that it imposes to go through all the entries of the multiples of each number during the sieving process. For instance, if the current entry corresponds to p, then any entry at locations $2p$, $3p$, $4p$ is changed to zero, and so on, until the stop criteria is reached, i.e., $p^2 > n$. The basic sieve of Eratosthenes proceeds in the same way on any other entry. It is easy to see that some numbers will be generated more than once, for example 6 is generated twice (from 2 and 3), and 12 is generated three times (from $2, 3$ and 4). The entries that are already zeros are left unchanged, but each entry must nevertheless be checked throughout the sieving process.

The main idea consists then in trying to prevent all numbers from being sieved "too many times". Sieving the multiples of any given number more than once must be avoided, as much as possible. All efficient sieving algorithms are based on similar techniques. So, the complexity $\mathcal{O}(n \ln \ln n)$ of the sieve of Eratosthenes may be somewhat improved by several clever arguments that are carried out by the above methods. Such sieve algorithms achieve a linear [12, 14, 20] or even a sublinear (step) complexity [14, 18]. So far, the best algorithm known is the "wheel sieve", designed in 1981 [18, 19]. It requires only $\mathcal{O}(n/\log\log n)$ steps to find the set of primes in the interval $[2,\ldots,n]$ (with $n > 4$), where each step is either for bookkeeping or an addition with integers at most n. Basically, the algorithm relies on the central result on the number of primes in arithmetic progressions. More precisely, Dirichlet's theorem states that if a, b are coprime integers ($gcd(a,b) := (a,b) = 1$) and $b > 0$, then the arithmetic progression $\{a, a+b, a+2b, \ldots\} = \{a \bmod (b)\}$ contains infinitely many primes [13, Thm. 15]. (See [8] for more details on the analysis of the wheel sieve algorithm.)

The present paper presents a new kind of fully distributed algorithm that finds all primes by sieving in a given interval $[1\ldots,n]$, using the properties of the wheel sieve using the SMER [19]. Some other distributed algorithms gen-

erating all prime numbers can be found in [6, 7], which use the properties of Dirichlet's theorem. In [17] another kind of distributed prime number generation is presented, based only on scheduling by multiple edge reverse framework [1].

In Sect. 2, the wheel sieve algorithm is introduced. In Sect. 3 and 4, the framework of the *scheduling by edge reversal* (SER) and the *scheduling by multiple edge reversal* (SMER) mechanisms are both introduced. Sect. 5 is devoted to the design of our distributed algorithm for sieving primes by using the SMER-based method applied to the wheel sieve. The worst-case complexity analysis of the algorithm is achieved in Sect. 6. The final Sect. 7 draws a short conclusion and offers some perspectives.

2 The Wheel Sieve

The wheel sieve derived from Pritchard's algorithm [18] operates basically by generating a set of numbers that are not multiples of the first k prime numbers. The sieve, applied on the resulting set from the wheel, eliminates the non prime numbers that remain in the set. This is the basic idea of the *wheel* which were employed as a reduced residue class $\bmod(\Pi_k)$, where Π_k denotes the product of the first k prime numbers [19]. $\mathcal{W}_k$ denotes the k-th wheel, which is defined as

$$\mathcal{R}(x) = \big\{x \ / \ 1 \leq y \leq x \text{ and } (y, x) = 1\big\}, \tag{1}$$

where $(x, y) := \gcd(x, y)$ is the greatest common divisor of the integers x and y.

The sieve introduced by the wheel sieve consists basically, after having generated the next wheel $\mathcal{W}_{k+1}$, in using the prime number $k+1$ to sieve the new wheel, generating all its multiples and removing them from $\mathcal{W}_{k+1}$. For more clarity this new set will denoted $\mathcal{S}_{k+1}$. It is clear that after $\mathcal{S}_{k+1}$ is obtained the algorithm proceeds to another sieving process, and eliminate the remaining composite numbers. The wheels are thus patterns that are repeated every Π_k times.

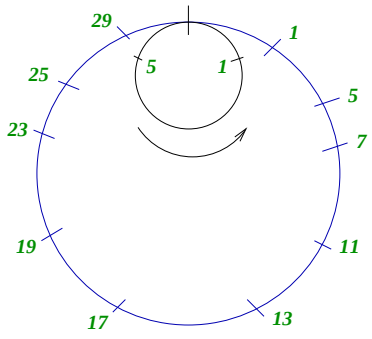

Fig. 1. Example of the generation of a wheel $\mathcal{W}_{k+1}$ starting from the preceding wheel $\mathcal{W}_k$

In Fig. 1 we use Π_2 in the first step of the wheel sieve as the product of the first two prime numbers (2 and 3) figured by the small circle; this generates all "pseudo-primes" numbers[4] between 1 and the new bound contained in the new wheel $\mathcal{W}_3 = \mathcal{R}(30)$, that is the actual bound Π_2 multiplied by the next prime $p_3 = 5$. The next prime is the first number after 1 which belongs to the interval being sieved [19] in the second wheel which contains now a value equal to 5.

Within the small wheel, Fig. 2 shows that all the pseudo-prime numbers of the big wheel $\{1, 5, 7, 11, 13, 17, 19, 23, 25, 29\}$ are generated.

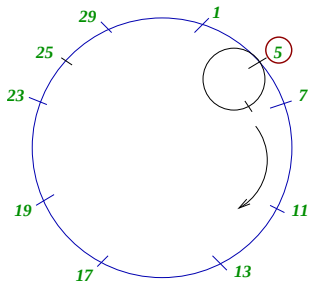

Fig. 2. Generation of the new "pseudo-prime" numbers

In Fig. 3 the process of generating the big wheel is going on, and the number 7 is generated, from the number 1 of the small wheel, which can be interpreted as if we were "rolling" the small circle inside the big one. This means that starting from a wheel $\mathcal{W}_k$, we can generate the next wheel $\mathcal{W}_{k+1}$ in a graphical way. The points where the elements of the wheel $\mathcal{W}_k$ touch the circle featuring $\mathcal{W}_{k+1}$ are the new pseudo-primes. More precisely, $\mathcal{W}_{k+1}$ is defined as

$$\mathcal{W}_{k+1} = \mathcal{W}_k \cup \left\{ x\Pi_k + y \;/\; x \in \{1, \ldots, p_{k+1} - 1\} \text{ and } y \in \mathcal{W}_k \right\}. \qquad (2)$$

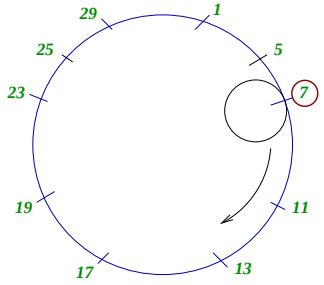

Fig. 3. Here we can see the generation of another new pseudo-prime (number 7)

[4] Numbers that are not multiples of the first k prime numbers.

Fig. 4 shows the final phase of the wheel sieve, where the multiples of the previous p_{k+1} (in that case, the number 5) are eliminated from the set $\mathcal{R}(\Pi_3)$. According to the definition of $\mathcal{W}_{k+1}$ in Eq. (2), we also define

$$\mathcal{S}_{k+1} = \mathcal{W}_{k+1} \setminus \{y \times p_{k+1} \ / \ y \in \mathcal{W}_{k+1}\}. \tag{3}$$

The previous wheel $\mathcal{W}_k$ is put in the center of the new wheel $\mathcal{S}_{k+1}$ (See Fig. 4). Then drawing a radius from the center of the small circle containing each pseudo-prime number of this circle, each one of the prolongations of such radii touches the big circle at every pseudo-prime that will be eliminated in the new wheel $\mathcal{W}_{k+1}$. Thus, the prime p_{k+1} will be put in the set $\mathcal{P}$ of all prime numbers.

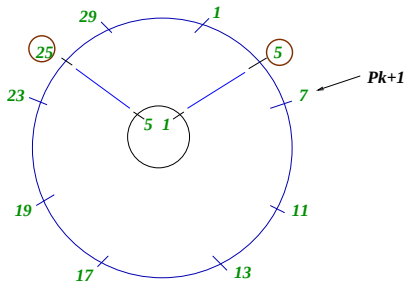

Fig. 4. The sieve being applied on the new wheel $\mathcal{W}_{k+1}$ to generate $\mathcal{S}_{k+1}$

In [16] a distributed version of the wheel sieve is proposed. It is implemented *via* a message passing interface specification (*lam-mpi 7.0.6 library*)[5] and the time measurements of a sequential and a distributed implementation of the wheel sieve are compared, together with a sequential and distributed implementation of the sieve of Eratosthenes. In [15] a fully distributed version of the wheel sieve is also presented.

3 Scheduling by Edge Reversal (SER)

Consider a neighbourhood-constrained system composed by a set of *processing elements* (PEs) and a set of *atomic shared resources* represented by a connected directed graph $G = (V, E)$, where V is the set of PEs and E the set of its directed edges (or arcs), stating the access topology (directed edges are henceforth refered to as arcs). The latter is defined in the following way: an arc exists between any two nodes *if and only if* the two corresponding PEs share at least one atomic resource. SER works as follows: starting from any acyclic orientation ω on G, there is at least one *sink* node, i.e., a node such that all its arcs are directed to itself; all sink nodes are allowed to operate while other nodes remain idle.

This obviously ensures mutual exclusion at any access made to shared resources by sink nodes. After operation, a sink node will reverse the orientation

of its arcs, becoming a *source* and thus releasing the access to resources to its neighbours. A new acyclic orientation is defined and the whole process is then repeated for the new set of sinks. Let $\tilde{\omega} = g(\omega)$ denote this greedy operation. SER can be regarded as the endless repetition of the application of $g(\omega)$ upon G.

Assuming that G is finite, it is easy to see that eventually a set of acyclic orientations will be repeated defining a period of length P. This simple dynamics ensures that no deadlocks or starvation will ever occur since in every acyclic orientation there exists at least one sink, i.e., one node allowed to operate. Also, it is proved that inside any period, every node operates exactly the same constant number of times (denoted M) [3].

SER is a fully distributed graph dynamics in which the sense of time is defined by its own operation, i.e., the synchronous behavior is equivalent to the case where every node in G takes an identical amount of time to operate and also an identical amount of time to reverse arcs. Another interesting observation to be made here is that any topology G will have its own set of possible SER dynamics [1].

As an example of SER's applicability, consider Dijkstra's paradigmatic Dining Philosophers problem [9] under heavy load, i.e., in the case philosophers are either "hungry" or "eating" (no "thinking" state). Such system can be represented by a set $\{P_1, \ldots, P_N\}$ of N PEs, in which each PE shares a resource both with its previous PE and its subsequent PE. Thus, taking the original configuration where $N = 5$ and setting an acyclic orientation over the 5 nodes ring, the resulting SER dynamics where $P = 5$ and $M = 2$ is illustrated in Fig. 5.

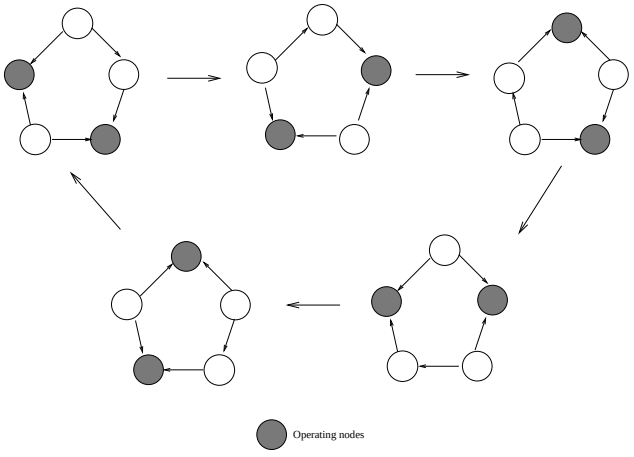

Fig. 5. SER dynamics for the Dining Philosophers under heavy load.

4 Scheduling by Multiple Edge Reversal (SMER)

SMER is a generalization of SER in which pre-specified access rates to atomic resources are imposed to processes in a distributed resource-sharing system represented by a multigraph $\mathcal{M} = (V, \mathcal{E})$. In contrast with SER, multiple edges can exist between any two nodes i and j ($i, j \in V$) in the SMER dynamics: there can exist $e_{i,j} \geq 0$ undirected edges connecting nodes i and j; such connected nodes are called "neighbours".

Let r_i denote the "reversibility" of node i, as defined in [2]. More precisely, reversibility r_i is the number of arcs that shall be reversed by i towards each of its neighbouring nodes at the end of each operation step (access to shared resources). Node i is called a r-sink if at least r_i arcs are directed to itself from each of its neighbours. In the SMER dynamics, each r-sink node i operates by reversing r_i arcs towards all of its neighbours, next a new set of r-sinks operates in turn, and so on. Similarly to sinks under SER, only r-sink nodes are allowed to operate under SMER. Unlike SER, nodes may operate more than once consecutively in SMER dynamics.

Let $\mu_0, \mu_1, \ldots$ be the sequence of orientations produced by SMER over $\mathcal{M}$ from the initial orientation μ_0. As infinite sequences are of our interest (originally motivated by the Dining Philosophers with rates (DPPr) problem [2]), let a_s^{ij} denote the greatest multiple of $\gcd(r_i, r_j)$ of r_i and r_j, which does not exceed the number of edges oriented from i to j in $\mu_s, s \geq 0$. Orientations μ_s, such that $f_{ij} = a_s^{ij} + a_s^{ji}$, $s \geq 0$, remaining constant as a consequence of the two terms changing by a certain multiple of $\gcd(r_i, r_j)$ (arcs reversed between neighbouring nodes i and j). Let $\mathcal{M}^{i,j}$ be the submultigraph of $\mathcal{M}$ induced by a pair of neighbouring nodes i and j. Moreover, let $\mu_0^{ij}, \mu_1^{ij}, \ldots$ the sequence of orientations of $\mathcal{M}^{i,j}$ produced by SMER from μ_0^{ij}. The following Lemma 1 states a basic topology constraint towards the definition of the multigraph $\mathcal{M}$.

Lemma 1. *([2, 10]) If* $max\{r_i, r_j\} \leq e_{i,j} \leq r_i + r_j - 1$, *aplication of SMER from* μ_0^{ij} *on* $\mathcal{M}^{i,j}$ *solves the instance of DPPr given by neighbouring nodes* i *and* j, r_i *and* r_j, *if and only if* $f_{ij} = r_i + r_j - \gcd(r_i, r_j)$. *In this case, the sequence* $\mu_0^{ij}, \mu_1^{ij}, \ldots \mu_s^{ij}$ *(*$s \geq 0$*) includes all orientations of* $\mathcal{M}^{ij}$ *that are legal for* i *and* j *given* μ_0^{ij}. *n a given arbitrary multigraph* $\mathcal{M}$. *If no deadlock arises for any initial orientation of the arcs between* i *and* j, *then*

It is important to know that there is always at least one SMER solution for any target system's topology having arbitrary pre-specified reversibilities at any of its nodes [10]. According to Lemma 1, since $e_{i,j} = r_i + r_j - 1$, either i or j is in a r-sink condition, independently of $\mu_s, s \geq 0$. It may also be seen that, between all pairs of neighbouring nodes i and j in $\mathcal{M}$, any SMER dynamics produces *one unique* period, given by the relation $P_{i,j} = (r_i + r_j)/\gcd(r_i, r_j)$ [2, 11]. This periodic property of SMER can be observed in Fig. 6,where $P_{i,j} = 8$ and the nodes in $\mathcal{M}$ share values that are pairwise coprime integers: such pairs (r_i, r_j) have no common divisors (but 1).

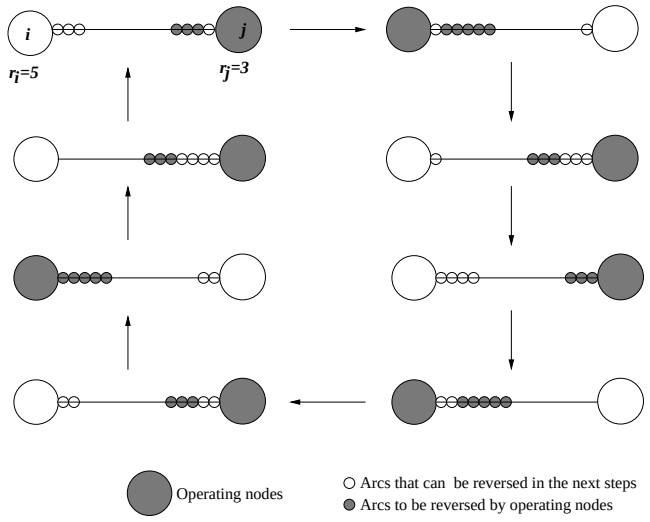

Fig. 6. An example of SMER, with period $P_{i,j} = 8$. Oriented arcs are represented by tokens.

5 The Distributed Wheel Sieve Algorithm using SMER

Let $\mathcal{M} = (V, \mathcal{E})$ be an arbitrary multigraph having N nodes. For the sake of simplicity, the distributed algorithm is actually assumed to sieve the restricted interval $\{2\} \cup \{\text{odd integers from } [3, \ldots, n]\}$, according to the parity of n. Such a SMER-based sieving algorithm is called *Semi-SMER*; this in contrast with the SMER dynamics described in Section 3, which considers the whole neighbourhood of any given node.

The procedure Semi-SMER is designed for any current node process $i \in V$, and it uses local variables, defined as follows:

- The interval I is set to an exclusive value in $\{1, 7, 11, 13, 17, 19, 23, 29\}$. For example, when we employ the third wheel $\mathcal{W}_3$ in the algorithm, the interval J is set to a value of Π_3 extended to $30, 60, 90$ ($2 \times 3 \times 5$, $2 \times 3^2 \times 5$ and $2 \times 3^3 \times 5$).
- $Neigh_i$ denotes the set of neighbours of process i, and the number of incoming arcs oriented from every $j \in Neigh_i$ to the current process i is denoted by the variable $incoming_i[j]$;
- $r_i[j]$ denotes the required number of arcs that shall be reversed by i towards every $j \in Neigh_i$, independently. Variable $r_i[j]$ takes its values in the interval I, and variable $r_j[i]$ takes its values in the interval J;
- $e_i[j]$ denotes the number of undirected edges (both outgoing and incoming arcs) connecting every pair of neighbours (i, j) in $\mathcal{M}$ (see Fig. 6);
- $a_i[j]$ denotes the number of incoming arcs oriented from each $j \in Neigh_i$ to i in the initial orientation;

– Process i also maintains the boolean variables $rev\_arc_i[j]$ and $end\_period\_i[j]$. If, at the end of the Semi-SMER period, $rev\_arc_i[j]$ is true for $j \in Neigh_i$, then $r_i[j]$ and $r_j[i]$ are coprime ($(r_i, rj) = 1$). The value of $end\_period\_i[j]$ checks whether the Semi-SMER between two nodes ended its execution or not;
– $PseudoPrimes$ contains the numbers generated by the extended wheel that consists in the remaining prime numbers.

Procedure WheelSieve-SMER(N)
var
$P_{i,j} = 0$ ($\star$ $P_{i,j}$ contains the size of the period of the SMER between two nodes $\star$)
$\mathcal{P}$; ($\star$ $\mathcal{P}$ is the set of the first k prime numbers $\star$)
$PseudoPrimes = 0$;
p_{k+1}; ($\star$ p_{k+1} is initialized with the next prime number $\star$)
$prime$: boolean **init** true;
$incoming_i[j]$: integer;
$rev\_arc_i[j]$: boolean **init** false;
$end\_period_i[j]$: boolean **init** false;
($\star$ $r_i[j]$ and $r_j[i]$ are initialized with the values of I and J, respectively. $\star$)
Begin
If $r_i[j] \leq r_j[i]$ **Then**
$a_i[j] = r_i[j]$;
$incoming_i[j] = r_i[j]$;
$e_i[j] = r_i[j] + r_j[i] - 1$;
Else
$a_i[j] = r_i[j] - 1$;
$incoming_i[j] = a_i[j]$;
$e_i[j] = r_i[j] + r_j[i] - 1$;
EndIf
While not $end\_period_i[j]$
If $incoming_i[j] \geq r_i[j]$
Then send message $\langle r_i[j] \rangle$ to $j \in Neigh_i$;
$incoming_i[j] = incoming_i[j] - r_i[j]$;
$P_{i,j} = P_{i,j} + 1$;
($\star$ The flipping arcs process is triggered $\star$)
Else
receive $\langle r_j[i] \rangle$ from $j \in Neigh_i$;
$r_i[j] = incoming_i[j] + r_j[i]$;
$P_{i,j} = P_{i,j} + 1$;
EndIf
If $incoming_i[j] = 0$ **Then** $rev\_arc_i[j] = true$;
EndIf
If $incoming_i[j] = a_i[j]$ **Then** $end\_period_i[j] = true$;

EndIf
EndWhile
$PseudoPrimes = P_{i,j}$;
$end\_period_i[j]$: boolean **init** false;
($\star$ $r_i[j]$ and $r_j[i]$ are initialized with the
values of p_{k+1} and $PseudoPrimes$, respectively. $\star$)
If $r_i[j] \leq r_j[i]$ **Then**
$a_i[j] = r_i[j]$;
$incoming_i[j] = r_i[j]$;
$e_i[j] = r_i[j] + r_j[i] - 1$;
Else
$a_i[j] = r_i[j] - 1$;
$incoming_i[j] = a_i[j]$;
$e_i[j] = r_i[j] + r_j[i] - 1$;
EndIf
While not $end\_period_i[j]$
If $incoming_i[j] \geq r_i[j]$
Then send message $\langle r_i[j] \rangle$ to $j \in Neigh_i$;
$incoming_i[j] = incoming_i[j] - r_i[j]$;
($\star$ The flipping arcs process is triggered $\star$)
Else
receive $\langle r_j[i] \rangle$ from $j \in Neigh_i$;
$r_i[j] = incoming_i[j] + r_j[i]$;
EndIf
If $incoming_i[j] = 0$ **Then** $rev\_arc_i[j] = true$;
EndIf
If $incoming_i[j] = a_i[j]$ **Then** $end\_period_i[j] = true$;
EndIf
EndWhile
If$rev\_arc_i[j] = true$;
Then $\mathcal{P} \cup P_{i,j}$;
EndIf
Return $\mathcal{P}$ ($\star$ $\mathcal{P}$ is the set of primes in the interval $[2 \ldots, n]$ $\star$)
end.

As pointed out, if we start initially the sieve with the third wheel there are eight processes, whose values are the numbers $\{1, 7, 11, 13, 17, 19, 23, 29\}$, that represent the values of I. The set J is started in accordance with N; for example, if $N = 230$, eight processes are needed, one for each value $\in J$. These values represent the multiples of $\Pi_3 = 30$. Beginning with these values (we consider that at the beginning, each process knows its identity), and after executing the WheelSieve-SMER, we obtain a set ($PseudoPrimes$) composed with the values of all periods spread in between the values of I and J. There remains the operation of sieving the $PseudoPrimes$ set with p_{k+1} in order to obtain $\mathcal{P}$.

6 Worst-Case complexity of the Algorithm

In order to sieve all primes from the interval $[2\ldots,n]$, the only fundamental operations explicitly used in the algorithm Semi-SMER are comparisons, additions and the sending and receiving of messages (arc reversals). Besides, a send-receive event and one comparison operation are assumed to take $\mathcal{O}(1)$ number of time slots.

The number of steps required by the algorithm is proportional to the period involved between any two nodes of $\mathcal{M}$ during the algorithm. Now, the largest period $P_{i,j}$ follows from Lemma 1 and [2]: $P_{i,j} = r_i + r_j$, when $(r_i, r_j) = 1$. Since $r_i \leq \lfloor\sqrt{n}\rfloor$ and $r_j \leq n$, for any pair of nodes (i,j), the procedure Semi-SMER(n) requires at most $n + \lfloor\sqrt{n}\rfloor$ steps.

Similarly, for any current pair of nodes (i,j) of $\mathcal{M}$ smaller than $\sqrt{n}$, the number of messages exchanged in the **while** loop is proportional to $P_{i,j} \times \deg_i$, with $\deg_i = \#Neigh_i$. Hence, if we let $P := \sup_{(i,j)\in V^2} P_{i,j}$ denote the largest period between all pairs $(i,j) \in V^2$, the maximum message complexity of the algorithm is proportional to $P \times \Delta_N$, where $1 \leq \Delta_N \leq N-1$ is the maximum multidegree of $\mathcal{M}$. Finally, the message complexity achieves at most $n\Delta_N + \lfloor\sqrt{n}\rfloor\Delta_N$. The maximum amount of memory space required per process is $\mathcal{O}(n)$ bits.

7 Conclusion and Perspectives

This paper introduced a totally new kind of SMER-based distributed sieve algorithm that generates all primes in a given interval $[2\ldots,n]$. Apart from observing that the fundamental operation of the Semi-SMER algorithm is a local comparison, it is also worth noticing that no *gcd* computation is needed. Moreover, no precomputation is assumed in the Semi-SMER complexity analysis (precomputation would take $\mathcal{O}\big(n \log\log \Pi_k\big)$, where Π_k denotes the product of the first k prime numbers, in the wheel sieve). This approach seems also general enough to compute some of the elementary arithmetic functions in number theory. For instance, *via* the gcd and inverse, the least common multiple of integers, and various basic multiplicative arithmetic functions, e.g. Euler's totient function $\phi(n)$, Möbius function $\mu(n)$ and divisor functions: $d(n)$, $\sigma(n)$, $\omega(n)$, $\Omega(n)$, etc.

Finally, it stems also from both computer-driven and theoretical results that the number of steps $T(n)$ executed by the algorithm stays always "very close" to the maximal number of steps. More precisely, $T(n) = n + \lfloor\sqrt{n}\rfloor - \varphi(n)$, where $\varphi(n)$ is a positive non periodic arithmetic function with rather small fluctuations when $n \geq 4$: we conjecture that $\varphi(n) < 5$ for "almost every" $n \geq 4$. Hence, for every $n \geq 4$, $\varphi(n)$ should yield an expected $\overline{\varphi(n)} = 2.47\ldots \pm \varepsilon_n$ for all $0 \leq \varepsilon_n < 1$, and the *average* number of steps required by the algorithm should then be expected to achieve $\overline{T(n)} \approx n + \lfloor\sqrt{n}\rfloor - 2.47\ldots$.

Acknowledgements

Support has been provided by the Brazilian agency Fundação Cearense de Apoio ao Desenvolvimento Científico e Tecnológico - FUNCAP.

References

1. V.C. Barbosa *An atlas of edge-reversal dynamics*, Research Notes in Mathematics, Chapman & Hall/CRC, Boca Raton, Florida, 2001.
2. V.C Barbosa, M.R.F. Benevides and F.M.G. França Sharing resources at nonuniform access rates, *Theory of Computing Systems* 34(1): 13–26, 2001.
3. V.C Barbosa and E. Gafni Concurrency in heavily loaded neighborhood-constrained systems, *ACM Transactions on Programming Languages and Systems* 11(4): 562–584, 1989.
4. S.H. Bokhari Multiprocessing the sieve of Eratosthenes, *IEEE Computer* 20(4): 50–58, 1987.
5. G. Burns, R. Daoud and J. Vaigl LAM: An Open Cluster Environment for MPI, *Proceedings of Supercomputing Symposium* pp. 379–386, 1994.
6. M. Cosnard and J.-L. Philippe Génération de nombres premiers en parallèle, *La lettre du transputer* pp. 3–12, 1989.
7. M. Cosnard and J.-L. Philippe Discovering new parallel algorithms. The sieve of Eratosthenes revisited, *Computer Algebra and Parallelism* pp. 1–18, 1989.
8. R. Crandall and C. Pomerance *Prime Numbers: a computational perspective*, Springer Verlag, 2001.
9. Edsger W. Dijkstra Hierarchical Ordering of Sequential Processes, *Acta Informatica* 1(2): 115–138, 1971.
10. F.M.G. França Scheduling weightless systems with self-timed boolean networks, *Workshop on Weightless Neural Networks* pp. 87–92, 1993.
11. F.M.G. França *Neural networks as neighbourhood-constrained systems*, PhD Thesis, Imperial College, London, 1994.
12. D. Gries and J. Misra A linear sieve algorithm for finding prime numbers, *Communications of the ACM* 21(12): 999—1003, 1978.
13. G. Hardy and E. Wright *An introduction to the theory of numbers*, Clarendon Press, Oxford, 1979.
14. H.G. Mairson Some new upper bounds on the generation of prime numbers, *Communication of the ACM* 20(9): 664–669, 1977.
15. G. Paillard A fully distributed prime numbers generation using the wheel sieve, *Parallel and Distributed Computing and networks* pp. 651–656, 2005.
16. G. Paillard and C. Lavault Le crible de la roue en distribué, *MAJECSTIC 2003 (MAnifestation des JEunes Chercheurs en STIC)* 2003.
17. G. Paillard, C. Lavault and F. França A SMER-based distributed prime sieving algorithm, *Technical Report 2004-04* LIPN, 2004.
18. P. Pritchard A sublinear additive sieve for finding prime numbers, *Communications of the ACM* 24(1): 18–23, 1981.
19. P. Pritchard Explaining the wheel sieve, *Acta Informatica* 17: 447–485, 1982.
20. J. Sorenson An introduction to prime numbers sieves, *Technical Report 909* Un. of Wisconsin, Computer Science Dept., 1990.
21. J. Sorenson and I. Parberry Two Fast Parallel Prime Number Sieves, *Information and Computation* pp. 115–130, 1994.